EXHIBIT B

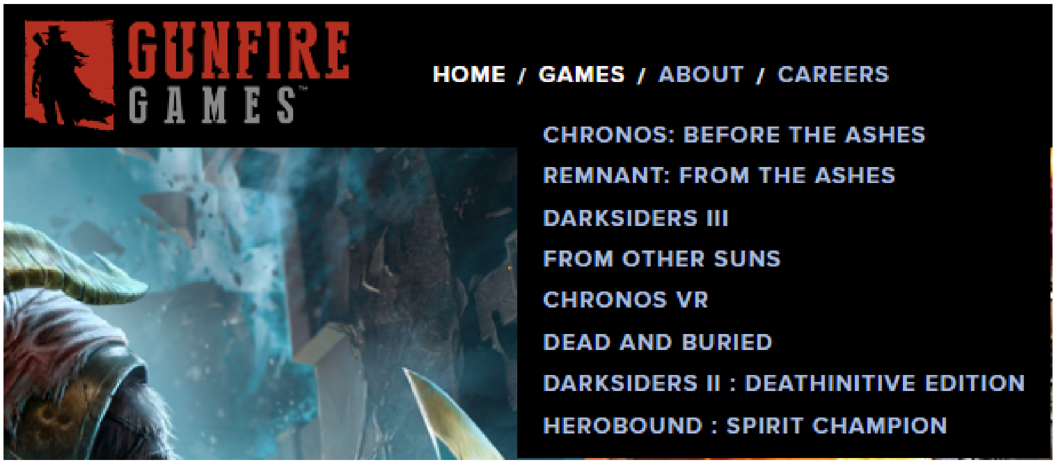
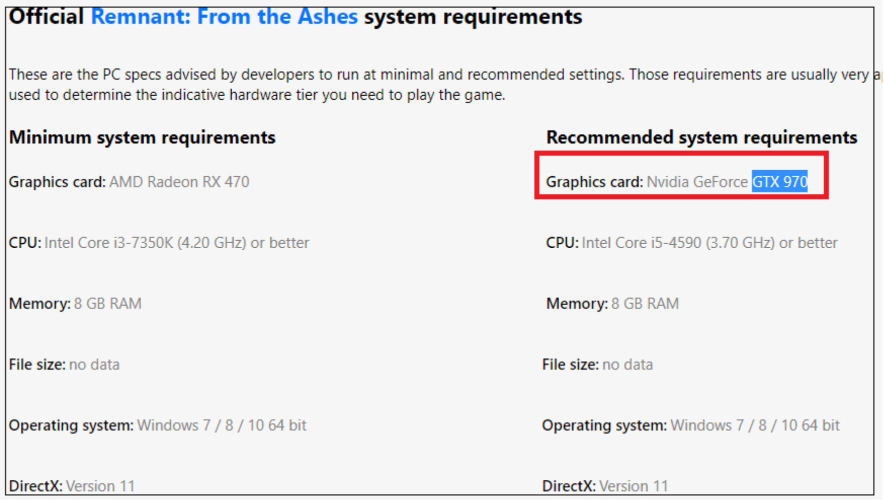
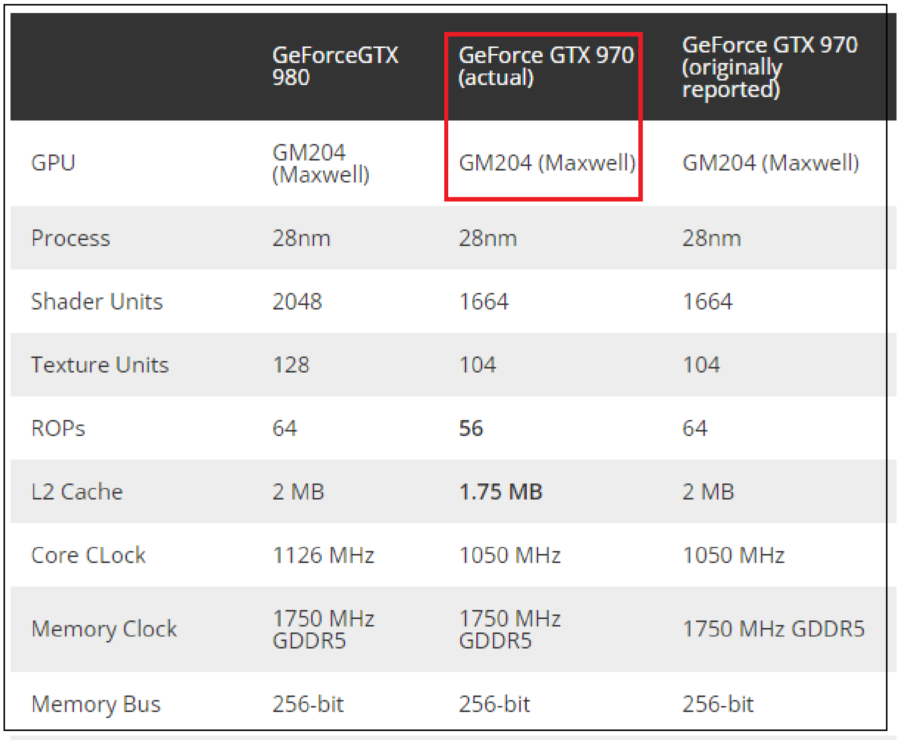
Claim	Analysis
[15.P] A method for organizing a computer system having a common display memory and main memory, comprising:	Gunfire Games Company has used many computers (including Accused Products), Such use includes internal use and use by its employees and contractors. Stoic Studio uses Accused Products for commercial purposes, such as developing software such as gaming softwares: Remnant, Chronos, Darksiders, that are executed on computers. The Accused Products include computers that include CPU and NVIDIA GPU (although Geforce GTX 970 is recommended, obviously, other advanced NVIDIA GPU such as GTX 1070 can also be used) components and other components.  GUNFIRE GAMES HOME / GAMES / ABOUT / CAREERS CHRONOS: BEFORE THE ASHES REMNANT: FROM THE ASHES DARKSIDERS III FROM OTHER SUNS CHRONOS VR DEAD AND BURIED DARKSIDERS II : DEATHINITIVE EDITION HEROBOUND : SPIRIT CHAMPION Source: http://gunfiregames.com/  **Official Remnant: From the Ashes system requirements** These are the PC specs advised by developers to run at minimal and recommended settings. Those requirements are usually very a used to determine the indicative hardware tier you need to play the game. **Minimum system requirements** **Graphics card:** AMD Radeon RX 470 **CPU:** Intel Core i3-7350K (4.20 GHz) or better **Memory:** 8 GB RAM **File size:** no data **Operating system:** Windows 7 / 8 / 10 64 bit **DirectX:** Version 11 **Recommended system requirements** **Graphics card:** Nvidia GeForce GTX 970 **CPU:** Intel Core i5-4590 (3.70 GHz) or better **Memory:** 8 GB RAM **File size:** no data **Operating system:** Windows 7 / 8 / 10 64 bit **DirectX:** Version 11 Source: https://technical.city/en/system-requirements/remnant-from-the-ashes

EXHIBIT B

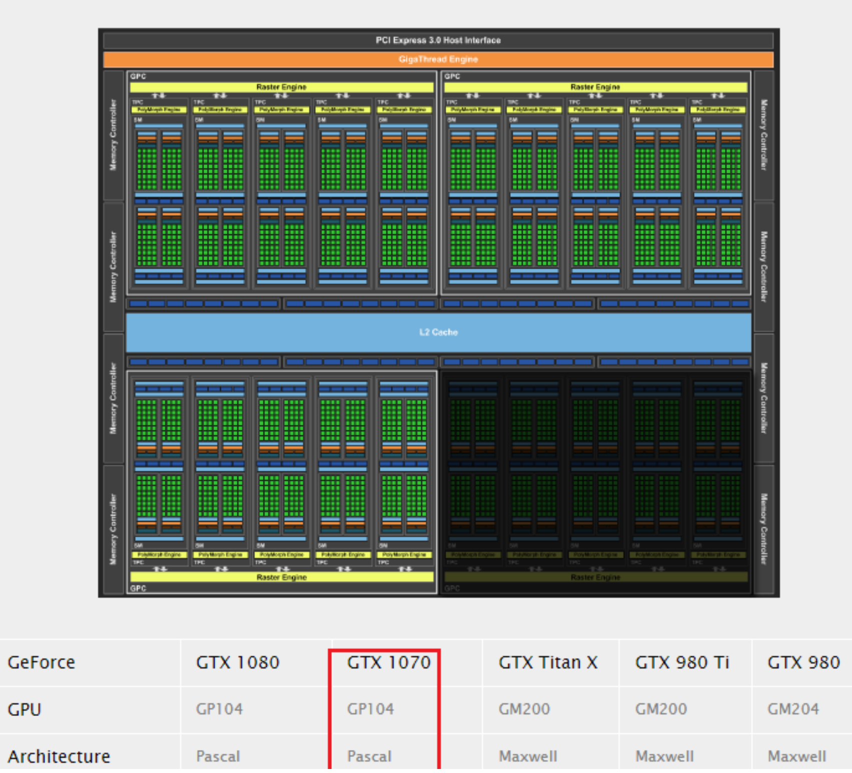
The GTX 970 operates on Maxwell architecture. and later NVIDIA develops GPUs based on Pascal architecture such as GTX 1070), the CPU and the GPU can simultaneously access managed memory, since they can both handle page faults. The GTX 1070 supports CUDA Unified Memory. Unified Memory ("common display memory and main memory") and virtual addressing allows sharing of memory between the CPU and the GPU and covers address space of CPUs (system memory), as well as the GPU's own memory.



	GeForceGTX 980	GeForce GTX 970 (actual)	GeForce GTX 970 (originally reported)
GPU	GM204 (Maxwell)	GM204 (Maxwell)	GM204 (Maxwell)
Process	28nm	28nm	28nm
Shader Units	2048	1664	1664
Texture Units	128	104	104
ROPs	64	**56**	64
L2 Cache	2 MB	**1.75 MB**	2 MB
Core CLock	1126 MHz	1050 MHz	1050 MHz
Memory Clock	1750 MHz GDDR5	1750 MHz GDDR5	1750 MHz GDDR5
Memory Bus	256-bit	256-bit	256-bit

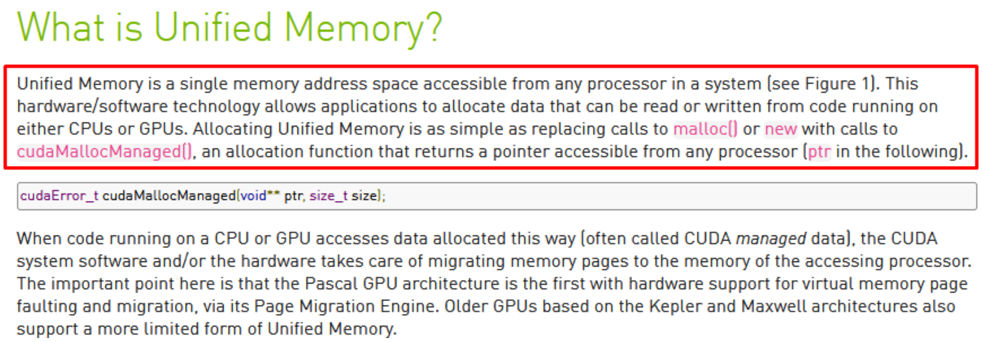
Source: https://www.tomshardware.com/news/nvidia-geforce-gtx-970-specifications,28464.html

EXHIBIT B



GeForce	GTX 1080	GTX 1070	GTX Titan X	GTX 980 Ti	GTX 980
GPU	GP104	GP104	GM200	GM200	GM204
Architecture	Pascal	Pascal	Maxwell	Maxwell	Maxwell

Source: https://www.guru3d.com/articles-pages/nvidia-geforce-gtx-1070-review,3.html



What is Unified Memory?

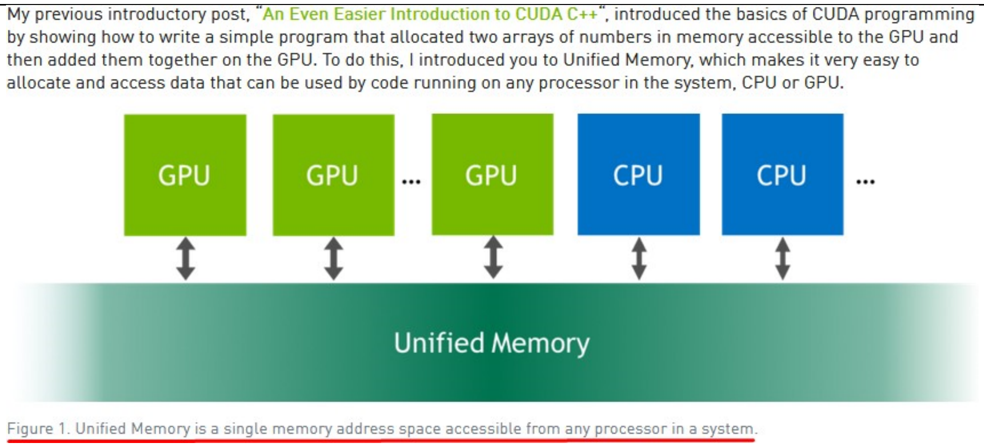
Unified Memory is a single memory address space accessible from any processor in a system (see Figure 1). This hardware/software technology allows applications to allocate data that can be read or written from code running on either CPUs or GPUs. Allocating Unified Memory is as simple as replacing calls to malloc() or new with calls to cudaMallocManaged(), an allocation function that returns a pointer accessible from any processor (ptr in the following).

```
cudaError_t cudaMallocManaged(void** ptr, size_t size);
```

When code running on a CPU or GPU accesses data allocated this way (often called CUDA *managed* data), the CUDA system software and/or the hardware takes care of migrating memory pages to the memory of the accessing processor. The important point here is that the Pascal GPU architecture is the first with hardware support for virtual memory page faulting and migration, via its Page Migration Engine. Older GPUs based on the Kepler and Maxwell architectures also support a more limited form of Unified Memory.

Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/

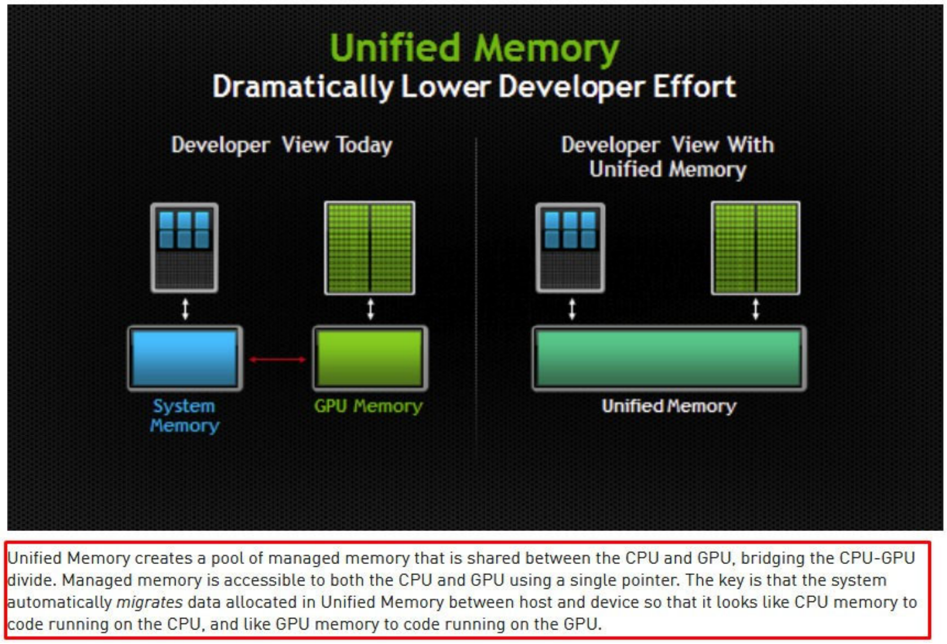
EXHIBIT B



My previous introductory post, "An Even Easier Introduction to CUDA C++", introduced the basics of CUDA programming by showing how to write a simple program that allocated two arrays of numbers in memory accessible to the GPU and then added them together on the GPU. To do this, I introduced you to Unified Memory, which makes it very easy to allocate and access data that can be used by code running on any processor in the system, CPU or GPU.

Figure 1. Unified Memory is a single memory address space accessible from any processor in a system.

Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/



Unified Memory

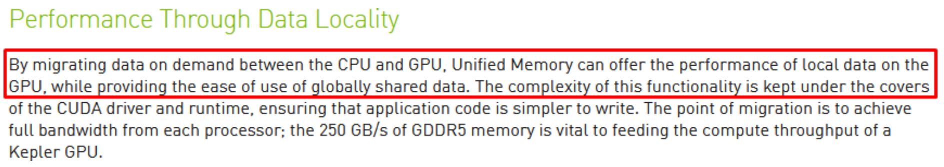
Dramatically Lower Developer Effort

Developer View Today

Developer View With Unified Memory

Unified Memory creates a pool of managed memory that is shared between the CPU and GPU, bridging the CPU-GPU divide. Managed memory is accessible to both the CPU and GPU using a single pointer. The key is that the system automatically *migrates* data allocated in Unified Memory between host and device so that it looks like CPU memory to code running on the CPU, and like GPU memory to code running on the GPU.

Source: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/



Performance Through Data Locality

By migrating data on demand between the CPU and GPU, Unified Memory can offer the performance of local data on the GPU, while providing the ease of use of globally shared data. The complexity of this functionality is kept under the covers of the CUDA driver and runtime, ensuring that application code is simpler to write. The point of migration is to achieve full bandwidth from each processor; the 250 GB/s of GDDR5 memory is vital to feeding the compute throughput of a Kepler GPU.

Source: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/

EXHIBIT B

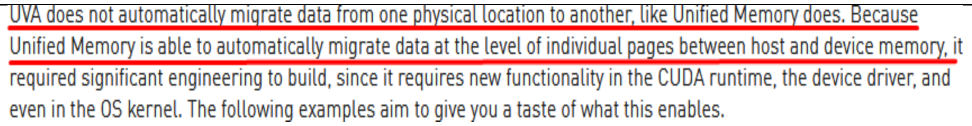
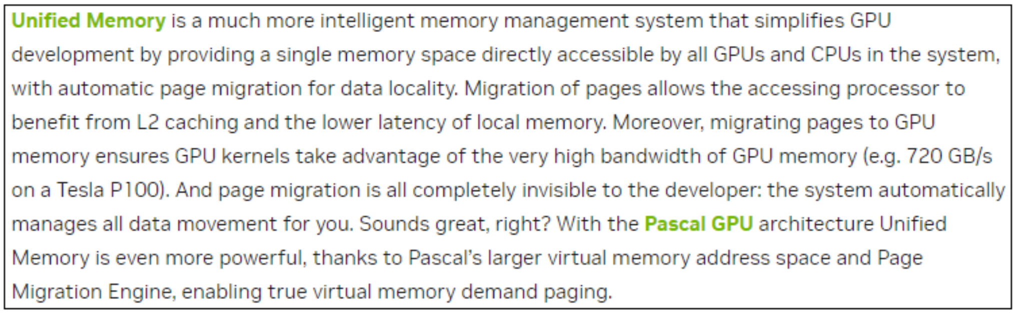
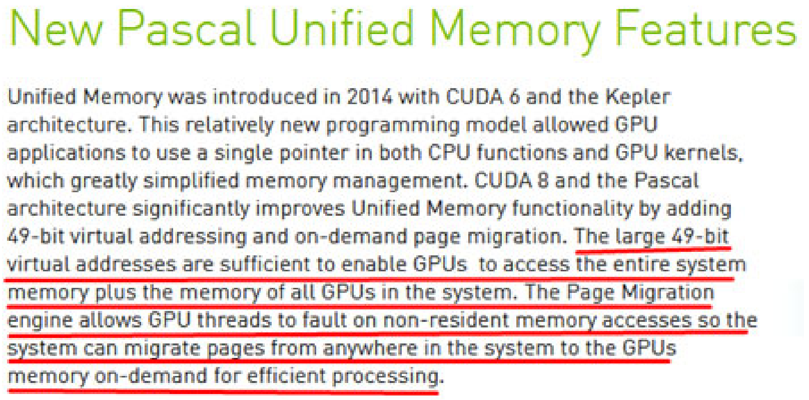
	 UVA does not automatically migrate data from one physical location to another, like Unified Memory does. Because Unified Memory is able to automatically migrate data at the level of individual pages between host and device memory, it required significant engineering to build, since it requires new functionality in the CUDA runtime, the device driver, and even in the OS kernel. The following examples aim to give you a taste of what this enables. Source: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/  **Unified Memory** is a much more intelligent memory management system that simplifies GPU development by providing a single memory space directly accessible by all GPUs and CPUs in the system, with automatic page migration for data locality. Migration of pages allows the accessing processor to benefit from L2 caching and the lower latency of local memory. Moreover, migrating pages to GPU memory ensures GPU kernels take advantage of the very high bandwidth of GPU memory (e.g. 720 GB/s on a Tesla P100). And page migration is all completely invisible to the developer: the system automatically manages all data movement for you. Sounds great, right? With the **Pascal GPU** architecture Unified Memory is even more powerful, thanks to Pascal's larger virtual memory address space and Page Migration Engine, enabling true virtual memory demand paging. Source: https://developer.nvidia.com/blog/beyond-gpu-memory-limits-unified-memory-pascal/  New Pascal Unified Memory Features Unified Memory was introduced in 2014 with CUDA 6 and the Kepler architecture. This relatively new programming model allowed GPU applications to use a single pointer in both CPU functions and GPU kernels, which greatly simplified memory management. CUDA 8 and the Pascal architecture significantly improves Unified Memory functionality by adding 49-bit virtual addressing and on-demand page migration. The large 49-bit virtual addresses are sufficient to enable GPUs to access the entire system memory plus the memory of all GPUs in the system. The Page Migration engine allows GPU threads to fault on non-resident memory accesses so the system can migrate pages from anywhere in the system to the GPUs memory on-demand for efficient processing. Source:https://developer.nvidia.com/blog/beyond-gpu-memory-limits-unified-memory-pascal
[15.1] providing a plurality of memory subsystems within said common display memory and main memory, each	The computer executing "Remnant" provides a plurality of memory subsystems within said common display memory and main memory, each memory subsystem having a dedicated memory channel.

EXHIBIT B

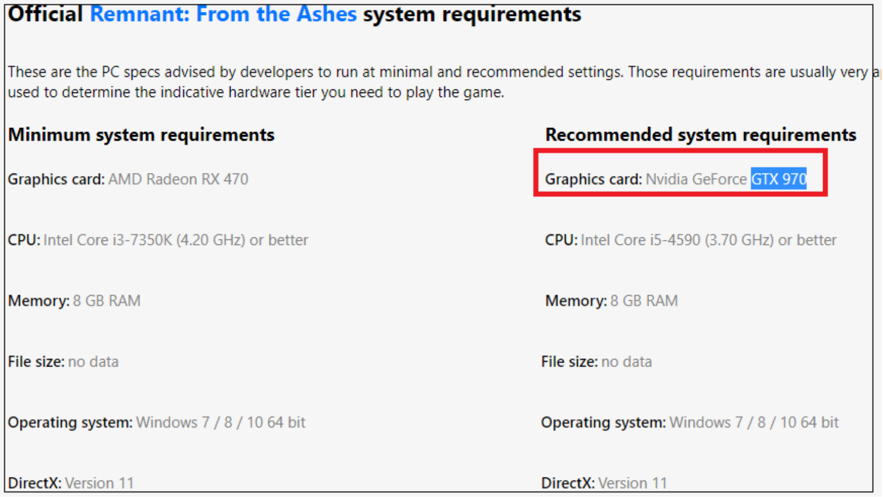
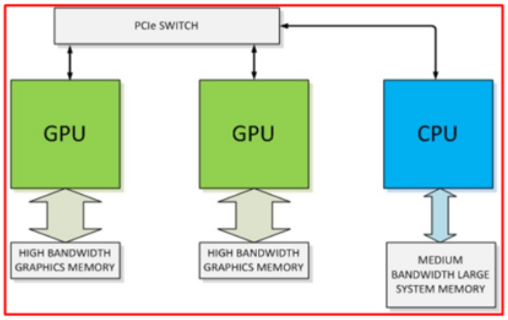
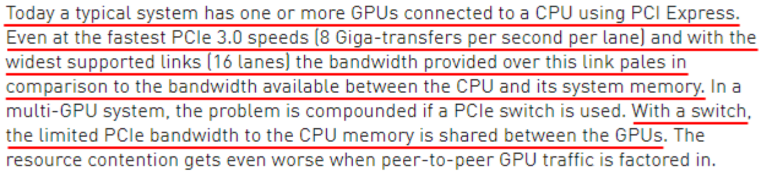
memory subsystem having a dedicated memory channel;	 **Official Remnant: From the Ashes system requirements** These are the PC specs advised by developers to run at minimal and recommended settings. Those requirements are usually very a used to determine the indicative hardware tier you need to play the game. **Minimum system requirements** **Graphics card:** AMD Radeon RX 470 **CPU:** Intel Core i3-7350K (4.20 GHz) or better **Memory:** 8 GB RAM **File size:** no data **Operating system:** Windows 7 / 8 / 10 64 bit **DirectX:** Version 11 **Recommended system requirements** **Graphics card:** Nvidia GeForce GTX 970 **CPU:** Intel Core i5-4590 (3.70 GHz) or better **Memory:** 8 GB RAM **File size:** no data **Operating system:** Windows 7 / 8 / 10 64 bit **DirectX:** Version 11 Source: https://technical.city/en/system-requirements/remnant-from-the-ashes  PCIe SWITCH GPU GPU CPU HIGH BANDWIDTH GRAPHICS MEMORY HIGH BANDWIDTH GRAPHICS MEMORY MEDIUM BANDWIDTH LARGE SYSTEM MEMORY  Today a typical system has one or more GPUs connected to a CPU using PCI Express. Even at the fastest PCIe 3.0 speeds (8 Giga-transfers per second per lane) and with the widest supported links (16 lanes) the bandwidth provided over this link pales in comparison to the bandwidth available between the CPU and its system memory. In a multi-GPU system, the problem is compounded if a PCIe switch is used. With a switch, the limited PCIe bandwidth to the CPU memory is shared between the GPUs. The resource contention gets even worse when peer-to-peer GPU traffic is factored in. Source: https://developer.nvidia.com/blog/nvlink-pascal-stacked-memory-feeding-appetite-big-data/

EXHIBIT B

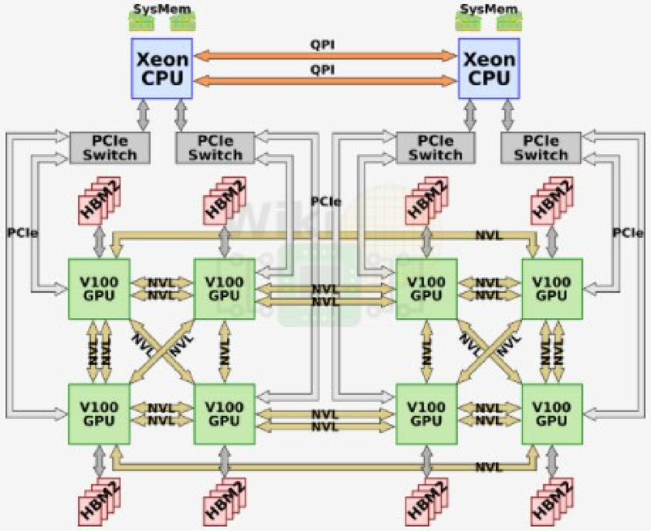
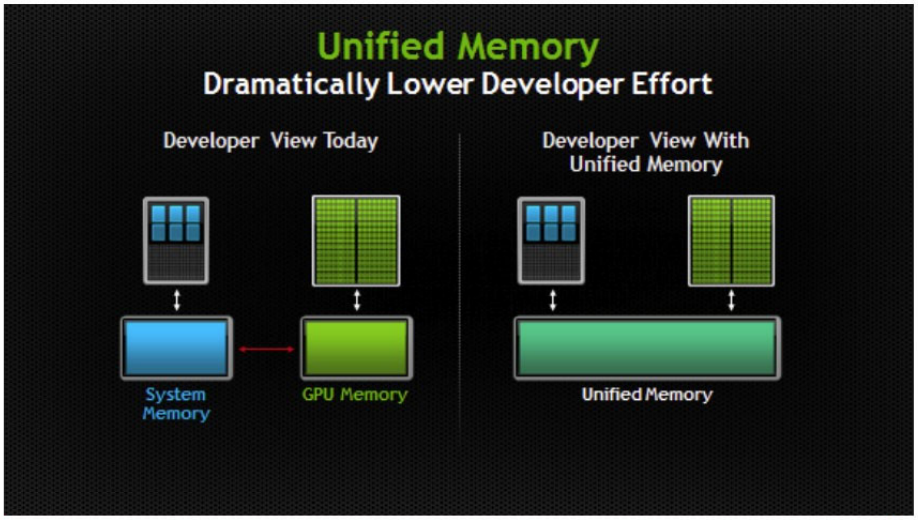
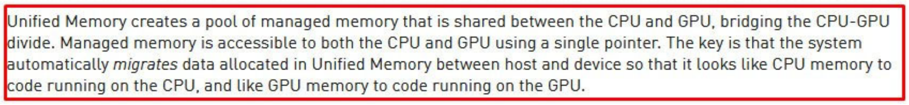
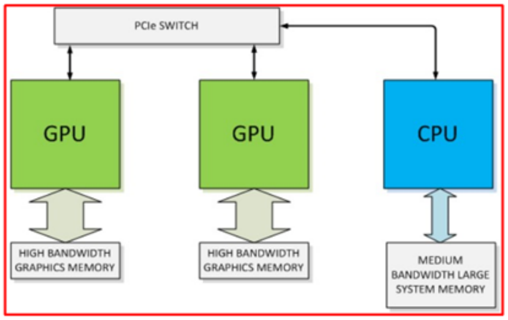
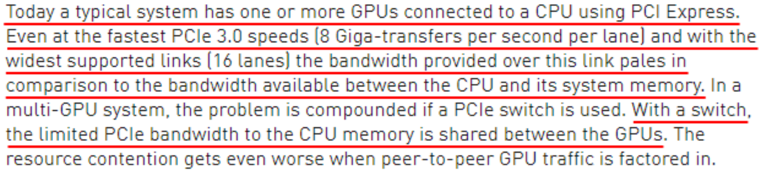
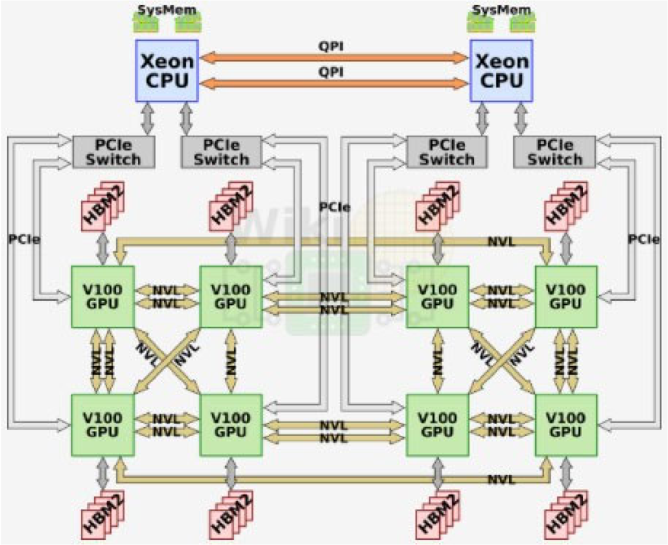
	 SysMem SysMem Xeon CPU Xeon CPU QPI QPI PCIe Switch PCIe Switch PCIe Switch PCIe Switch PCIe PCIe PCIe HBM2 V100 GPU NVL Source: https://fuse.wikichip.org/news/1224/a-look-at-nvidias-nvlink-interconnect-and-the-nvswitch/  Unified Memory Dramatically Lower Developer Effort Developer View Today Developer View With Unified Memory  Unified Memory creates a pool of managed memory that is shared between the CPU and GPU, bridging the CPU-GPU divide. Managed memory is accessible to both the CPU and GPU using a single pointer. The key is that the system automatically *migrates* data allocated in Unified Memory between host and device so that it looks like CPU memory to code running on the CPU, and like GPU memory to code running on the GPU. Source: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/
[15.2] providing a memory channel data switch and controller (DSC) unit coupled to each of said	The NVIDIA Pascal architecture utilizes the PCIe Switch ("memory channel data switch and controller (DSC)") that enables Unified Memory access by the CPU or the GPUs using the dedicated memory channels. The Pascal architecture also supports the NVlink switch. In addition, a computer that includes a CPU and a GPU includes switching circuitry and control as needed to support unified memory (which is used in NVIDIA GPUs with the Pascal architecture).

EXHIBIT B

<table>
<tr>
<td>memory subsystems through said dedicated memory channels; and</td>
<td>

PCIe SWITCH
GPU
GPU
CPU
HIGH BANDWIDTH GRAPHICS MEMORY
HIGH BANDWIDTH GRAPHICS MEMORY
MEDIUM BANDWIDTH LARGE SYSTEM MEMORY


Today a typical system has one or more GPUs connected to a CPU using PCI Express. Even at the fastest PCIe 3.0 speeds (8 Giga-transfers per second per lane) and with the widest supported links (16 lanes) the bandwidth provided over this link pales in comparison to the bandwidth available between the CPU and its system memory. In a multi-GPU system, the problem is compounded if a PCIe switch is used. With a switch, the limited PCIe bandwidth to the CPU memory is shared between the GPUs. The resource contention gets even worse when peer-to-peer GPU traffic is factored in.

Source: https://developer.nvidia.com/blog/nvlink-pascal-stacked-memory-feeding-appetite-big-data/



Source: https://fuse.wikichip.org/news/1224/a-look-at-nvidias-nvlink-interconnect-and-the-nvswitch/
</td>
</tr>
</table>

EXHIBIT B

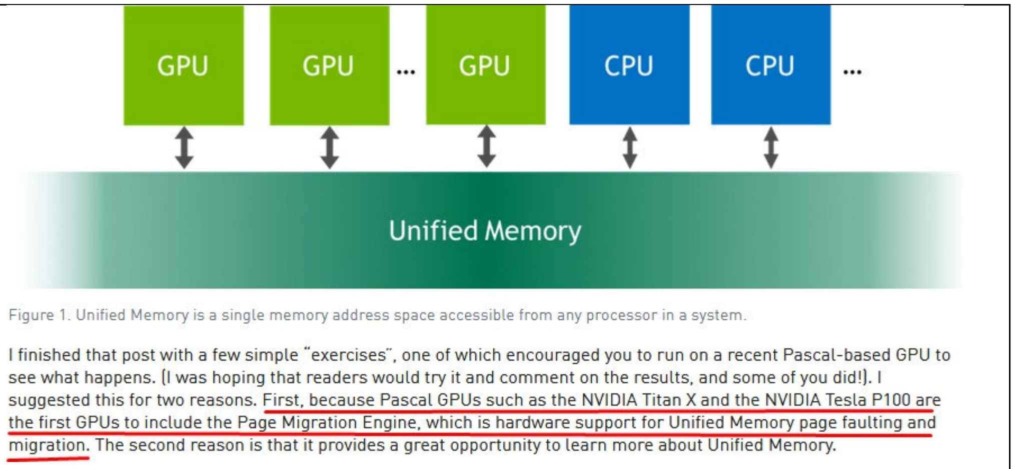
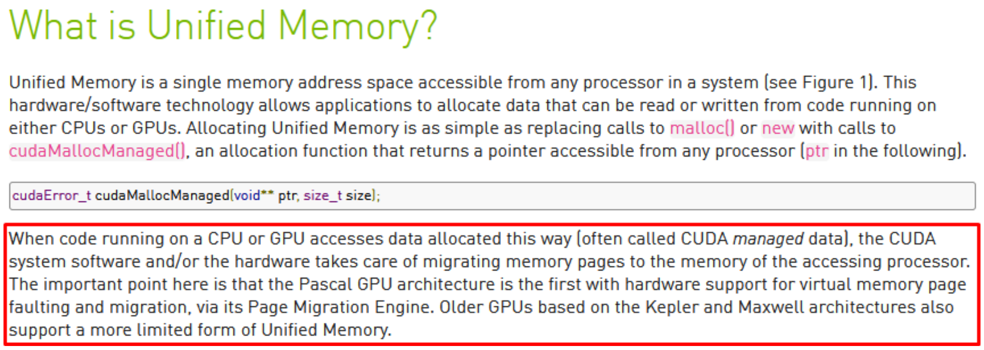


Figure 1. Unified Memory is a single memory address space accessible from any processor in a system.

I finished that post with a few simple "exercises", one of which encouraged you to run on a recent Pascal-based GPU to see what happens. (I was hoping that readers would try it and comment on the results, and some of you did!). I suggested this for two reasons. First, because Pascal GPUs such as the NVIDIA Titan X and the NVIDIA Tesla P100 are the first GPUs to include the Page Migration Engine, which is hardware support for Unified Memory page faulting and migration. The second reason is that it provides a great opportunity to learn more about Unified Memory.

Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/



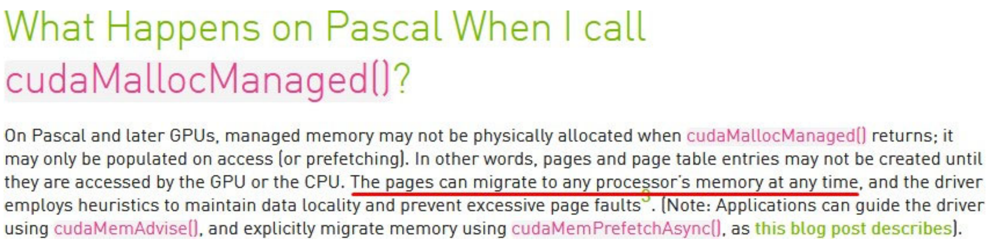
What is Unified Memory?

Unified Memory is a single memory address space accessible from any processor in a system (see Figure 1). This hardware/software technology allows applications to allocate data that can be read or written from code running on either CPUs or GPUs. Allocating Unified Memory is as simple as replacing calls to malloc() or new with calls to cudaMallocManaged(), an allocation function that returns a pointer accessible from any processor (ptr in the following).

```
cudaError_t cudaMallocManaged(void** ptr, size_t size);
```

When code running on a CPU or GPU accesses data allocated this way (often called CUDA *managed* data), the CUDA system software and/or the hardware takes care of migrating memory pages to the memory of the accessing processor. The important point here is that the Pascal GPU architecture is the first with hardware support for virtual memory page faulting and migration, via its Page Migration Engine. Older GPUs based on the Kepler and Maxwell architectures also support a more limited form of Unified Memory.

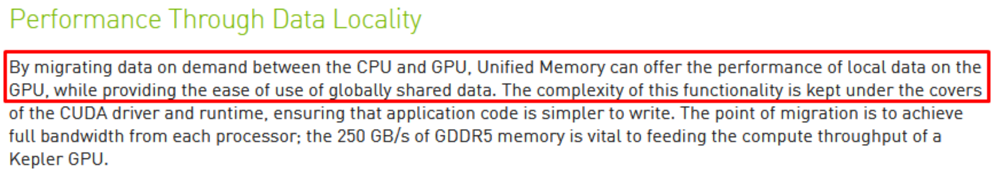
Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/



What Happens on Pascal When I call cudaMallocManaged()?

On Pascal and later GPUs, managed memory may not be physically allocated when cudaMallocManaged() returns; it may only be populated on access (or prefetching). In other words, pages and page table entries may not be created until they are accessed by the GPU or the CPU. The pages can migrate to any processor's memory at any time, and the driver employs heuristics to maintain data locality and prevent excessive page faults[5]. (Note: Applications can guide the driver using cudaMemAdvise(), and explicitly migrate memory using cudaMemPrefetchAsync(), as this blog post describes).

Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/



Performance Through Data Locality

By migrating data on demand between the CPU and GPU, Unified Memory can offer the performance of local data on the GPU, while providing the ease of use of globally shared data. The complexity of this functionality is kept under the covers of the CUDA driver and runtime, ensuring that application code is simpler to write. The point of migration is to achieve full bandwidth from each processor; the 250 GB/s of GDDR5 memory is vital to feeding the compute throughput of a Kepler GPU.

Source: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/

EXHIBIT B

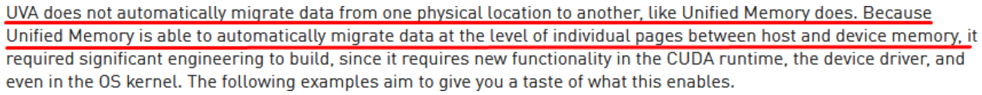
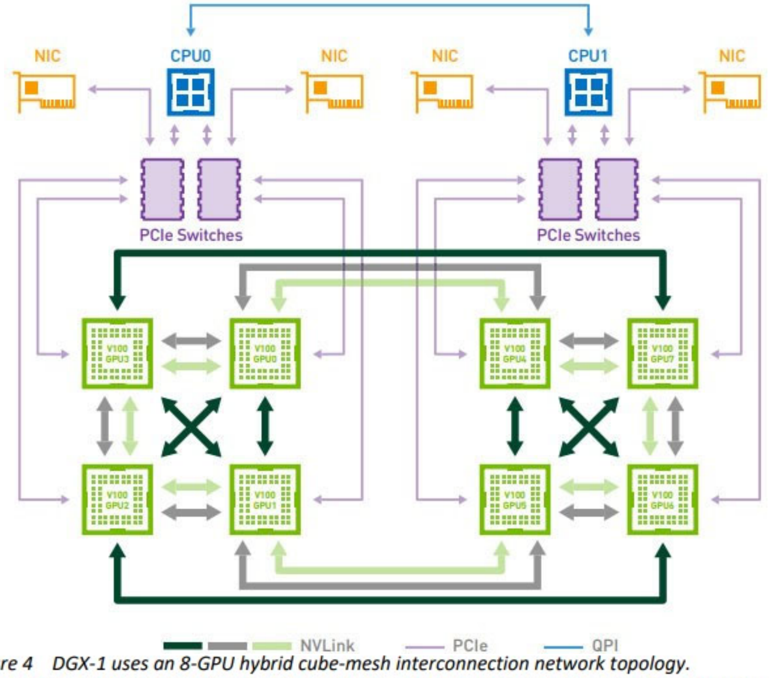
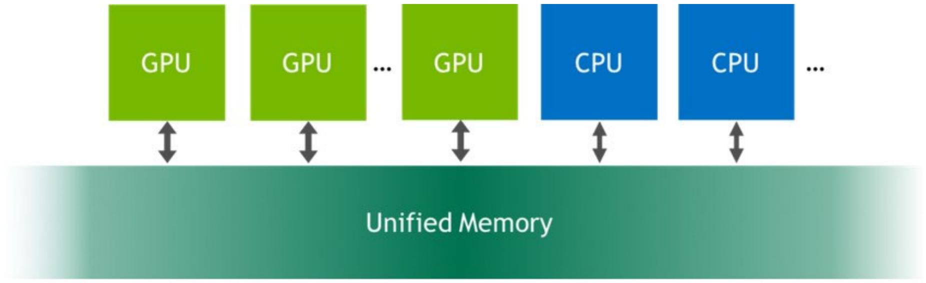
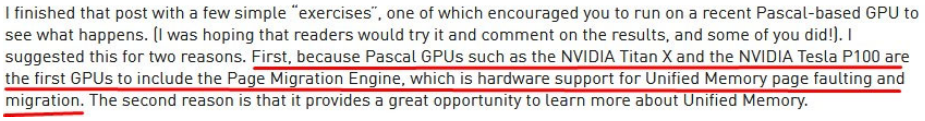
	 UVA does not automatically migrate data from one physical location to another, like Unified Memory does. Because Unified Memory is able to automatically migrate data at the level of individual pages between host and device memory, it required significant engineering to build, since it requires new functionality in the CUDA runtime, the device driver, and even in the OS kernel. The following examples aim to give you a taste of what this enables. Source: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/
[15.3] providing a plurality of processor and/or peripheral subsystems including a graphics/drawing and display (GDD) subsystem and a central processing unit (CPU) subsystem controller unit, each of said processor and/or peripheral subsystems coupled to said memory channel data switch and controller unit;	The computer executing "Remnant" include a CPU and at least on GPU. The PCIe Switch is coupled to each of the GPU and the CPU such that data (pages) is migrated between the CPU and the GPU.  NIC CPU0 NIC NIC CPU1 NIC PCIe Switches PCIe Switches V100 GPU3 V100 GPU0 V100 GPU4 V100 GPU7 V100 GPU2 V100 GPU1 V100 GPU5 V100 GPU6 NVLink PCIe QPI *Figure 4 DGX-1 uses an 8-GPU hybrid cube-mesh interconnection network topology. The corners of the mesh-connected faces of the cube are connected to the PCIe tree network, which also connects to the CPUs and NICs.* Source: https://images.nvidia.com/content/pdf/dgx1-v100-system-architecture-whitepaper.pdf Page 11  GPU GPU ... GPU CPU CPU ... Unified Memory Figure 1. Unified Memory is a single memory address space accessible from any processor in a system.  I finished that post with a few simple "exercises", one of which encouraged you to run on a recent Pascal-based GPU to see what happens. (I was hoping that readers would try it and comment on the results, and some of you did!). I suggested this for two reasons. First, because Pascal GPUs such as the NVIDIA Titan X and the NVIDIA Tesla P100 are the first GPUs to include the Page Migration Engine, which is hardware support for Unified Memory page faulting and migration. The second reason is that it provides a great opportunity to learn more about Unified Memory. Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/

EXHIBIT B

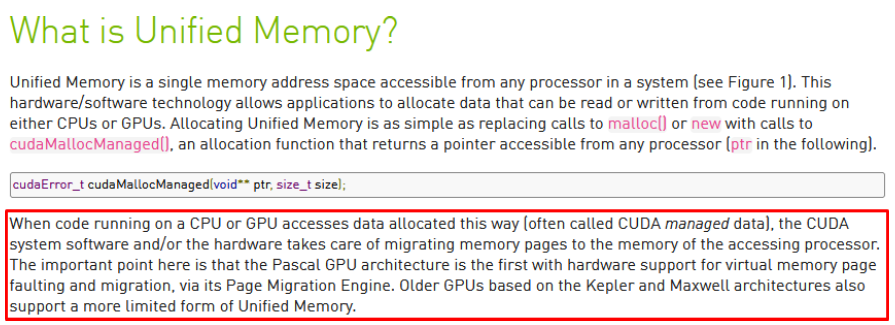
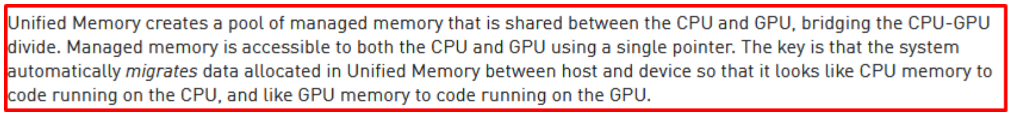
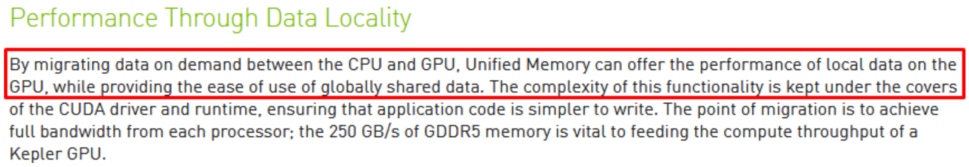
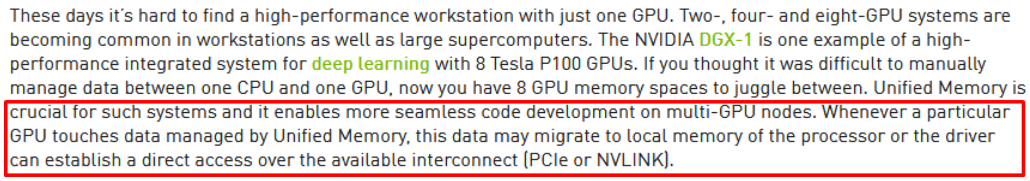
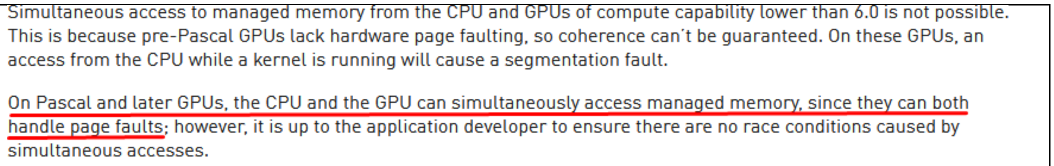
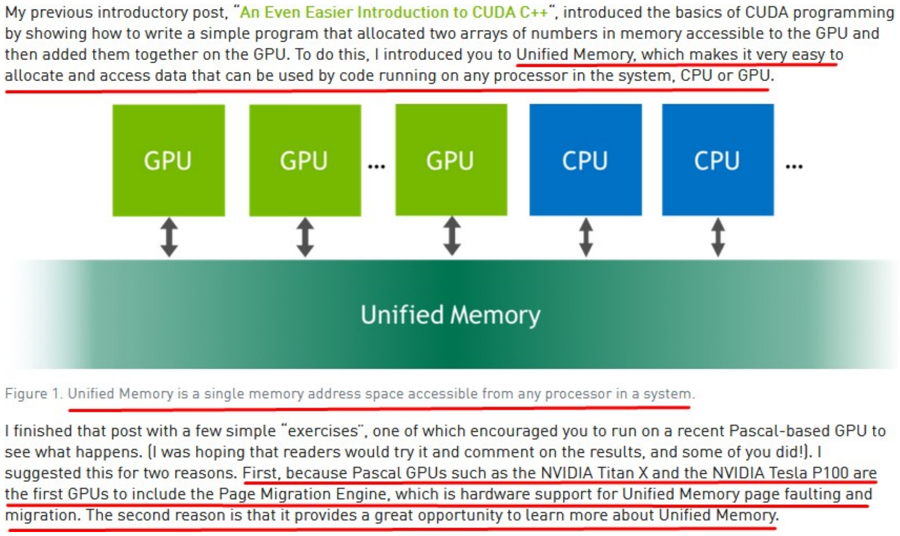
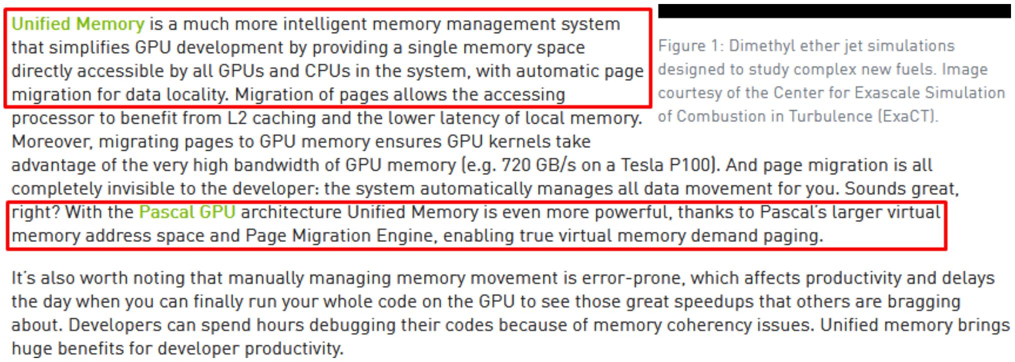
	 ## What is Unified Memory? Unified Memory is a single memory address space accessible from any processor in a system (see Figure 1). This hardware/software technology allows applications to allocate data that can be read or written from code running on either CPUs or GPUs. Allocating Unified Memory is as simple as replacing calls to malloc() or new with calls to cudaMallocManaged(), an allocation function that returns a pointer accessible from any processor (ptr in the following). `cudaError_t cudaMallocManaged(void** ptr, size_t size);` When code running on a CPU or GPU accesses data allocated this way (often called CUDA *managed* data), the CUDA system software and/or the hardware takes care of migrating memory pages to the memory of the accessing processor. The important point here is that the Pascal GPU architecture is the first with hardware support for virtual memory page faulting and migration, via its Page Migration Engine. Older GPUs based on the Kepler and Maxwell architectures also support a more limited form of Unified Memory. Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/  Unified Memory creates a pool of managed memory that is shared between the CPU and GPU, bridging the CPU-GPU divide. Managed memory is accessible to both the CPU and GPU using a single pointer. The key is that the system automatically *migrates* data allocated in Unified Memory between host and device so that it looks like CPU memory to code running on the CPU, and like GPU memory to code running on the GPU. Source: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/  ### Performance Through Data Locality By migrating data on demand between the CPU and GPU, Unified Memory can offer the performance of local data on the GPU, while providing the ease of use of globally shared data. The complexity of this functionality is kept under the covers of the CUDA driver and runtime, ensuring that application code is simpler to write. The point of migration is to achieve full bandwidth from each processor; the 250 GB/s of GDDR5 memory is vital to feeding the compute throughput of a Kepler GPU. Source: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/  These days it's hard to find a high-performance workstation with just one GPU. Two-, four- and eight-GPU systems are becoming common in workstations as well as large supercomputers. The NVIDIA DGX-1 is one example of a high-performance integrated system for deep learning with 8 Tesla P100 GPUs. If you thought it was difficult to manually manage data between one CPU and one GPU, now you have 8 GPU memory spaces to juggle between. Unified Memory is crucial for such systems and it enables more seamless code development on multi-GPU nodes. Whenever a particular GPU touches data managed by Unified Memory, this data may migrate to local memory of the processor or the driver can establish a direct access over the available interconnect (PCIe or NVLINK). Source: https://developer.nvidia.com/blog/beyond-gpu-memory-limits-unified-memory-pascal/
[15.4] wherein said GDD subsystem can access any one of aid plurality of memory subsystems and said CPU subsystem	The computer executing "Remnant" include a CPU and NVIDIA GPU. The Nvidia GPU is based on NVIDIA's Pascal architecture and supports CUDA 6 Unified Memory. Unified Memory and virtual addressing allows sharing of memory between the CPU and the GPU and covers address space of the CPU, as well as the GPU's own memory. The CPU and the GPU can concurrently access the Unified Memory that includes multiple memory subsystems.

EXHIBIT B

controller unit can concurrently access any other of said plurality of memory subsystems through said memory channel DSC unit.	 Simultaneous access to managed memory from the CPU and GPUs of compute capability lower than 6.0 is not possible. This is because pre-Pascal GPUs lack hardware page faulting, so coherence can't be guaranteed. On these GPUs, an access from the CPU while a kernel is running will cause a segmentation fault. On Pascal and later GPUs, the CPU and the GPU can simultaneously access managed memory, since they can both handle page faults; however, it is up to the application developer to ensure there are no race conditions caused by simultaneous accesses. Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/  My previous introductory post, "An Even Easier Introduction to CUDA C++", introduced the basics of CUDA programming by showing how to write a simple program that allocated two arrays of numbers in memory accessible to the GPU and then added them together on the GPU. To do this, I introduced you to Unified Memory, which makes it very easy to allocate and access data that can be used by code running on any processor in the system, CPU or GPU. GPU GPU ... GPU CPU CPU ... Unified Memory Figure 1. Unified Memory is a single memory address space accessible from any processor in a system. I finished that post with a few simple "exercises", one of which encouraged you to run on a recent Pascal-based GPU to see what happens. (I was hoping that readers would try it and comment on the results, and some of you did!). I suggested this for two reasons. First, because Pascal GPUs such as the NVIDIA Titan X and the NVIDIA Tesla P100 are the first GPUs to include the Page Migration Engine, which is hardware support for Unified Memory page faulting and migration. The second reason is that it provides a great opportunity to learn more about Unified Memory. Source: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/  Unified Memory is a much more intelligent memory management system that simplifies GPU development by providing a single memory space directly accessible by all GPUs and CPUs in the system, with automatic page migration for data locality. Migration of pages allows the accessing processor to benefit from L2 caching and the lower latency of local memory. Moreover, migrating pages to GPU memory ensures GPU kernels take advantage of the very high bandwidth of GPU memory (e.g. 720 GB/s on a Tesla P100). And page migration is all completely invisible to the developer: the system automatically manages all data movement for you. Sounds great, right? With the Pascal GPU architecture Unified Memory is even more powerful, thanks to Pascal's larger virtual memory address space and Page Migration Engine, enabling true virtual memory demand paging. Figure 1: Dimethyl ether jet simulations designed to study complex new fuels. Image courtesy of the Center for Exascale Simulation of Combustion in Turbulence (ExaCT). It's also worth noting that manually managing memory movement is error-prone, which affects productivity and delays the day when you can finally run your whole code on the GPU to see those great speedups that others are bragging about. Developers can spend hours debugging their codes because of memory coherency issues. Unified memory brings huge benefits for developer productivity. Source: https://developer.nvidia.com/blog/beyond-gpu-memory-limits-unified-memory-pascal/